EMB0000243c3415

**시스템프로그래밍 2020 보고서**

**보고서 제출서약서**

**나는 숭실대학교 소프트웨어학부의 일원으로 명예를 지키면서 생활하고 있습니다.**

**나는 보고서를 작성하면서 다음과 같은 사항을 준수하였음을 엄숙히 서약합니다.**

**1. 나는 자력으로 보고서를 작성하였습니다.**

**1.1. 나는 동료의 보고서를 베끼지 않았습니다.**

**1.2. 나는 비공식적으로 얻은 해답/해설을 기초로 보고서를 작성하지 않았습니다.**

**2. 나는 보고서에서 참조한 문헌의 출처를 밝혔으며 표절하지 않았습니다. (나는 특히**

**인터넷에서 다운로드한 내용을 보고서에 거의 그대로 복사하여 사용하지 않았습니다.)**

**3. 나는 보고서를 제출하기 전에 동료에게 보여주지 않았습니다.**

**4. 나는 보고서의 내용을 조작하거나 날조하지 않았습니다.**

|  |  |
| --- | --- |
| **과목** | **시스템프로그래밍 2020** |
| **과제명** | **프로젝트 1** |
| **담당교수** | **최 재 영 교 수** |
| **제출인** | ***소프트웨어학부 20150286 최유준* (출석번호 310번)** |
| **제출일** | **2020년 4월 27일** |

**-------------------- 차 례 --------------------**

1. **동기/목적**
2. **설계/구현 아이디어**

**2.1 설계 아이디어**

**2.2 구현 아이디어**

1. **수행결과**
2. **결론 및 보충할 점**
3. **소스코드**

**1장 동기/목적**

이번 시스템 프로그래밍의 프로젝트는 주어진 input을 기계어로 바꾸는 어셈블리 프로그램을 완성하는 것이었습니다. 저는 이 프로젝트를 손수 진행하면서 시스템 프로그래밍에 대한 이해도를 높이고 이론적으로 학습했던 내용을 실제로 구현해 보았습니다. 결론적으로 프로그램을 완성하면서 프로그램에 대한 이해도를 얻을 수 있었습니다. 다만 프로젝트에서 주어진 input만이 아니라 완벽한 어셈블러를 구현하려면 더 많은 노력이 필요함을 알 수 있었습니다.

**2장 설계/구현 아이디어**

**2.1 설계 아이디어**

추가된 프로그램의 실행 순서는 다음과 같습니다.

1. assem\_pass1 함수 호출
   1. input\_data의 line\_num 만큼 token\_parsing 함수 반복 호출
      1. 입력받은 input\_data를 분석 및 분할하여 각각의 token\_table에 저장
   2. token\_table의 각 line에 locator값을 정해주고 label이 symbol일 경우 sym\_table에, literal일 경우 literal\_table에 label과 locator값 저장
2. make\_symtab\_output 함수 호출
3. make\_literaltab\_output 함수 호출
4. assem\_pass2 함수 호출

4.1 각 section 별로 token\_table을 호출하여 각 인자의의 opcode, nixbpe, assem\_pass1에서 저장한 locator값을 이용하여 objectcode를 저장

1. make\_objectcode\_output 함수 호출
   1. control section별로 분리
      1. 각 section별로 section name, start address, section length 출력
      2. EXTDEF와 EXTREF의 유무를 검사하여 출력
      3. text의 길이가 30이 넘지 않도록 반복하며 objectcode를 출력
      4. EXTREF 인자에 해당하는 string을 Modify 출력

**\*모듈별 설계 아이디어**

1. **assem\_pass1**

token\_table symbol\_table

symbol addr

label operator operand..

literal\_table

literal addr

token\_table->label을 분석하여 symbol일 경우 symbol\_table에, literal일 경우 literal\_table에 저장한다.

1. **make\_symtab\_output**

sym\_table에서 인자를 하나씩 받아서 각각의 symbol과 address를 출력한다.

1. **make\_literaltab\_output**

literal\_table에서 인자를 하나씩 받아서 각각의 literal과 address를 출력한다.

1. **assem\_pass2**

token\_table의 각 인자들을 분석하여 기계어로 바꾼 후 저장해준다.

1. **make\_objectcode\_output**

assem\_pass2에서 기계어로 바꾼 내용을 섹션별로 출력해준다.

**2.2 구현 아이디어**

1. assem\_pass1

assem\_pass1에서 각 Section별로 token\_table을 호출한다. token\_table의 label을 조사하여 label이 있을 경우 해당 라벨을 sym\_table.symbol에 저장한다. 각 인자별로 operator가 n형식일 경우 locctr 변수에 +n을 해주어 위치를 저장한다. 이후 sym\_table.addr에 저장한다.

1. make\_symtab\_output

sym\_line만큼 for문으로 sym\_table을 반복하여 출력한다.

1. make\_literaltab\_output

literal\_line만큼 for문으로 literal\_table을 반복하여 출력한다

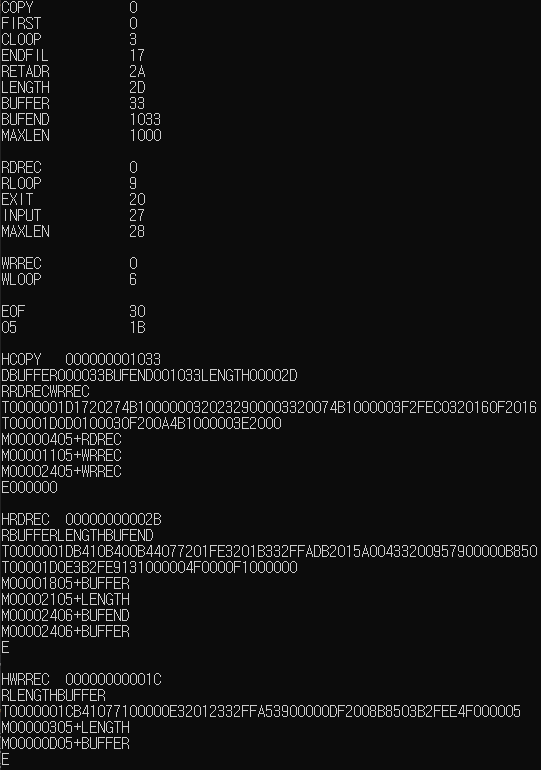
1. assem\_pass2

각 Section별로 token\_table을 호출한다. operator를 검사하여 Instruction일 경우 opcode를 찾아낸다. 각 line의 format과 addressing mode를 검사하여 비트연산을 통해 nixbpe의 값을 저장한다. 각 line의 addressing mode와 operand를 통하여 address를 계산한 후 opcode, nixbpe와 합쳐서 objectcode배열에 저장한다.

1. make\_objectcode\_output

각 Section별로 H, D, R, T, M, E로 구분하여 출력한다.

**3장 수행결과**

****

표준출력 output

사진이(가) 표시된 사진

자동 생성된 설명스크린샷이(가) 표시된 사진

자동 생성된 설명

스크린샷이(가) 표시된 사진

자동 생성된 설명

file output

**4장 결론 및 보충할 점**

어셈블러를 만드는 과정은 많은 조건분기와 예외처리 등이 필요한 작업인 만큼 많은 내용을 실제적으로 학습할 수 있는 과정이었습니다. 이 과정을 통해 실제로 구동할 수 있는 어셈블러를 만들 프로젝트도 기대해볼 수 있을 것으로 생각합니다.

이 프로젝트를 만드는 과정에서 많은 예외처리와 조건분기 등이 일반적인 무작위 input을 예상한 것이 아닌, 프로젝트에서 주어진 input을 토대로 감안하여 프로그램을 만들다보니 이 프로그램을 모든 input data에 범용적으로 사용할 수는 없게 되었습니다. 이는 아직 시스템 프로그래밍이 익숙치 않고 어셈블러를 완벽하게 구현할 정도의 skill을 갖추지 못했기 때문이며 추후 일반적인 input data에서 범용성있게 작동할 수 있는 assembler로 개선할 필요가 있습니다.

* **DEBUGING**

스크린샷, 검은색, 컴퓨터, 모니터이(가) 표시된 사진

자동 생성된 설명

**5장 소스코드**

**my\_assembler\_20150286.h**

*/\**

*\* my\_assembler 함수를 위한 변수 선언 및 매크로를 담고 있는 헤더 파일이다.*

*\**

*\*/*

#define MAX\_INST 256

#define MAX\_LINES 5000

#define MAX\_OPERAND 3

*/\**

*\* instruction 목록 파일로 부터 정보를 받아와서 생성하는 구조체 변수이다.*

*\* 구조는 각자의 instruction set의 양식에 맞춰 직접 구현하되*

*\* 라인 별로 하나의 instruction을 저장한다.*

*\*/*

**struct** inst\_unit

{

char opname[6];

int format;

char opcode[2];

int odnum;

*/\* add your code here \*/*

};

*// instruction의 정보를 가진 구조체를 관리하는 테이블 생성*

**typedef** **struct** inst\_unit inst;

inst \*inst\_table[MAX\_INST];

int inst\_index;

*/\**

*\* 어셈블리 할 소스코드를 입력받는 테이블이다. 라인 단위로 관리할 수 있다.*

*\*/*

char \*input\_data[MAX\_LINES];

**static** int line\_num;

*/\**

*\* 어셈블리 할 소스코드를 토큰단위로 관리하기 위한 구조체 변수이다.*

*\* operator는 renaming을 허용한다.*

*\* nixbpe는 8bit 중 하위 6개의 bit를 이용하여 n,i,x,b,p,e를 표시한다.*

*\*/*

**struct** token\_unit

{

char \*label; *//명령어 라인 중 label*

char \*operator; *//명령어 라인 중 operator*

char \*operand[MAX\_OPERAND]; *//명령어 라인 중 operand*

char \*comment; *//명령어 라인 중 comment*

char nixbpe; *// 추후 프로젝트에서 사용된다.*

};

**typedef** **struct** token\_unit token;

token \*token\_table[MAX\_LINES];

**static** int token\_line;

**static** int tok\_adr[MAX\_LINES];*//각 line마다 주소를 저장하는 변수*

**static** int objectcode[MAX\_LINES];*//명령어를 저장하는 변수*

**static** int csect\_len[3];*//각각의 section마다 길이를 저장하는 변수*

**static** int start = 1;*//각 section에서 반복문 시작할 인덱스를 저장하는 함수*

**static** int print[MAX\_LINES];*//출력배열*

**static** int mod[3] = { 0 };*//Modify를 위한 배열*

*/\**

*\* 심볼을 관리하는 구조체이다.*

*\* 심볼 테이블은 심볼 이름, 심볼의 위치로 구성된다.*

*\* 추후 프로젝트에서 사용된다.*

*\*/*

**struct** symbol\_unit

{

char symbol[10];

int addr;

};

**typedef** **struct** symbol\_unit symbol;

symbol sym\_table[MAX\_LINES];

**static** int sym\_line;

*/\**

*\* 리터럴을 관리하는 구조체이다.*

*\* 리터럴 테이블은 리터럴의 이름, 리터럴의 위치로 구성된다.*

*\* 추후 프로젝트에서 사용된다.*

*\*/*

**struct** literal\_unit

{

char literal[10];

int addr;

};

**typedef** **struct** literal\_unit literal;

literal literal\_table[MAX\_LINES];

**static** int locctr;*//현재위치(tok\_adr찾기위해)*

**static** int lit\_line;*//literal line수*

*//--------------*

**static** char \*input\_file;

**static** char \*output\_file;

int init\_my\_assembler(void);

int init\_inst\_file(char \*inst\_file);

int init\_input\_file(char \*input\_file);

int token\_parsing(char \*str);

int search\_opcode(char \*str);

**static** int assem\_pass1(void);

void make\_opcode\_output(char \*file\_name);

*/\* 추후 프로젝트에서 사용하게 되는 함수\*/*

void make\_symtab\_output(char \*file\_name);

void make\_literaltab\_output(char \*file\_name);

**static** int assem\_pass2(void);

void make\_objectcode\_output(char \*file\_name);

**my\_assembler\_20150286.c**

*/\**

*\* 화일명 : my\_assembler\_00000000.c*

*\* 설 명 : 이 프로그램은 SIC/XE 머신을 위한 간단한 Assembler 프로그램의 메인루틴으로,*

*\* 입력된 파일의 코드 중, 명령어에 해당하는 OPCODE를 찾아 출력한다.*

*\* 파일 내에서 사용되는 문자열 "00000000"에는 자신의 학번을 기입한다.*

*\*/*

*/\**

*\**

*\* 프로그램의 헤더를 정의한다.*

*\**

*\*/*

#include *<stdio.h>*

#include *<stdlib.h>*

#include *<string.h>*

#include *<fcntl.h>*

*// 파일명의 "00000000"은 자신의 학번으로 변경할 것.*

#include *"my\_assembler\_20150286.h"*

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 사용자로 부터 어셈블리 파일을 받아서 명령어의 OPCODE를 찾아 출력한다.*

*\* 매계 : 실행 파일, 어셈블리 파일*

*\* 반환 : 성공 = 0, 실패 = < 0*

*\* 주의 : 현재 어셈블리 프로그램의 리스트 파일을 생성하는 루틴은 만들지 않았다.*

*\* 또한 중간파일을 생성하지 않는다.*

*\* ----------------------------------------------------------------------------------*

*\*/*

int main(int args, char \*arg[])

{

**if** (init\_my\_assembler() < 0)

{

printf("init\_my\_assembler: 프로그램 초기화에 실패 했습니다.**\n**");

**return** -1;

}

**if** (assem\_pass1() < 0)

{

printf("assem\_pass1: 패스1 과정에서 실패하였습니다. **\n**");

**return** -1;

}

*//make\_opcode\_output("output\_20150286.txt");*

make\_symtab\_output("symtab\_20150286");

make\_literaltab\_output("literaltab\_20150286");

**if**(assem\_pass2() < 0 ){

printf(" assem\_pass2: 패스2 과정에서 실패하였습니다. **\n**") ;

**return** -1 ;

}

make\_objectcode\_output("output\_20150286") ;

**return** 0;

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 프로그램 초기화를 위한 자료구조 생성 및 파일을 읽는 함수이다.*

*\* 매계 : 없음*

*\* 반환 : 정상종료 = 0 , 에러 발생 = -1*

*\* 주의 : 각각의 명령어 테이블을 내부에 선언하지 않고 관리를 용이하게 하기*

*\* 위해서 파일 단위로 관리하여 프로그램 초기화를 통해 정보를 읽어 올 수 있도록*

*\* 구현하였다.*

*\* ----------------------------------------------------------------------------------*

*\*/*

int init\_my\_assembler(void)

{

int result;

**if** ((result = init\_inst\_file("inst.data")) < 0)

**return** -1;

**if** ((result = init\_input\_file("input.txt")) < 0)

**return** -1;

**return** result;

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 머신을 위한 기계 코드목록 파일을 읽어 기계어 목록 테이블(inst\_table)을*

*\* 생성하는 함수이다.*

*\* 매계 : 기계어 목록 파일*

*\* 반환 : 정상종료 = 0 , 에러 < 0*

*\* 주의 : 기계어 목록파일 형식은 자유롭게 구현한다. 예시는 다음과 같다.*

*\**

*\* ===============================================================================*

*\* | 이름 | 형식 | 기계어 코드 | 오퍼랜드의 갯수 | NULL|*

*\* ===============================================================================*

*\**

*\* ----------------------------------------------------------------------------------*

*\*/*

int init\_inst\_file(char \*inst\_file)

{

FILE \*file;

int errno;

*/\* add your code here \*/*

file = fopen(inst\_file, "r");

int i = 0;

**if**(file != NULL) {

errno = 0;

**while** (!feof(file)) {

inst\_table[i] = (inst\*)malloc(**sizeof**(inst));*//동적할당*

**if** (4 != fscanf(file, "%s %d %s %d", inst\_table[i]->opname, &inst\_table[i]->format, &inst\_table[i]->opcode, &inst\_table[i]->odnum)) {

errno = -1;

**break**;

}

**else** {*//*

i++;

inst\_index++;

}

}

}

**return** errno;

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 어셈블리 할 소스코드를 읽어 소스코드 테이블(input\_data)를 생성하는 함수이다.*

*\* 매계 : 어셈블리할 소스파일명*

*\* 반환 : 정상종료 = 0 , 에러 < 0*

*\* 주의 : 라인단위로 저장한다.*

*\**

*\* ----------------------------------------------------------------------------------*

*\*/*

int init\_input\_file(char \*input\_file)

{

FILE \*file;

int errno;

*/\* add your code here \*/*

int i = 0;

char str[500];

int length;

file = fopen(input\_file, "r");

**if** (file != NULL) {

**while** (!feof(file)) {

fgets(str, **sizeof**(str), file);

str[strlen(str) - 1] = '\0';

length = strlen(str);

input\_data[i] = (char \*)malloc(**sizeof**(char)\*length+1);

**if** (str[0] == '.')

**continue**;

strcpy(input\_data[i], str);

i++;

++line\_num;

}

errno = 0;

}

**else**

errno = 1;

**return** errno;

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 소스 코드를 읽어와 토큰단위로 분석하고 토큰 테이블을 작성하는 함수이다.*

*\* 패스 1로 부터 호출된다.*

*\* 매계 : 파싱을 원하는 문자열*

*\* 반환 : 정상종료 = 0 , 에러 < 0*

*\* 주의 : my\_assembler 프로그램에서는 라인단위로 토큰 및 오브젝트 관리를 하고 있다.*

*\* ----------------------------------------------------------------------------------*

*\*/*

int token\_parsing(char \*str)

{

*/\* add your code here \*/*

char check;

int first;

*//label이 없으면 first = 0, 있으면 first = 1*

sscanf(str, "%c", &check);

**if** ((check == '\t') || (check == ' '))

first = 0;

**else**

first = 1;

token\_table[token\_line] = (token\*)malloc(**sizeof**(token));*//동적할당*

**if** (check == '.') {

**return** 0;

}

token\_table[token\_line]->operand[0] = "";

token\_table[token\_line]->operand[1] = "";

token\_table[token\_line]->operand[2] = ""; *//operand 초기화*

**if** (first == 0) { *//label이 없을 때*

token\_table[token\_line]->label = "";

token\_table[token\_line]->operator = strtok(str, "**\t**");

token\_table[token\_line]->operand[0] = strtok(NULL, "**\t**");

token\_table[token\_line]->comment = strtok(NULL, "**\n**");

}

**else** **if** (first == 1) { *//label이 있을 때*

token\_table[token\_line]->label = strtok(str, "**\t**");

token\_table[token\_line]->operator = strtok(NULL, "**\t**");

token\_table[token\_line]->operand[0] = strtok(NULL, "**\t**");

token\_table[token\_line]->comment = strtok(NULL, "**\n**");

}

*//operand 분할*

token\_table[token\_line]->operand[0] = strtok(token\_table[token\_line]->operand[0], ",");

token\_table[token\_line]->operand[1] = strtok(NULL, ",");

token\_table[token\_line]->operand[2] = strtok(NULL, ",");

token\_line++;

**return** 0;

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 입력 문자열이 기계어 코드인지를 검사하는 함수이다.*

*\* 매계 : 토큰 단위로 구분된 문자열*

*\* 반환 : 정상종료 = 기계어 테이블 인덱스, 에러 < 0*

*\* 주의 :*

*\**

*\* ----------------------------------------------------------------------------------*

*\*/*

int search\_opcode(char \*str)

{

int e = 0;*//4형식인지*

**if** (str == NULL)

**return** 0;

**if** (str[0] == '+') {

e = 1;*//4형식*

str = str + 1;

}

**for** (int i = 0; i < inst\_index; i++) {*//inst\_table에서 명령어 검색*

**if** (strcmp(inst\_table[i]->opname, str) == 0) {

**return** i;

}

}

**return** -1;

*/\* add your code here \*/*

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 어셈블리 코드를 위한 패스1과정을 수행하는 함수이다.*

*\* 패스1에서는..*

*\* 1. 프로그램 소스를 스캔하여 해당하는 토큰단위로 분리하여 프로그램 라인별 토큰*

*\* 테이블을 생성한다.*

*\**

*\* 매계 : 없음*

*\* 반환 : 정상 종료 = 0 , 에러 = < 0*

*\* 주의 : 현재 초기 버전에서는 에러에 대한 검사를 하지 않고 넘어간 상태이다.*

*\* 따라서 에러에 대한 검사 루틴을 추가해야 한다.*

*\**

*\* -----------------------------------------------------------------------------------*

*\*/*

**static** int assem\_pass1(void)

{

*/\* add your code here \*/*

*//char\* str;*

int i;

int len;

int temaddr;

**for** (i = 0; i < line\_num; i++) {

token\_parsing(input\_data[i]);

}

sym\_line = 0;

i = 0;

locctr = 0;

len = 0;

**while** (strcmp(token\_table[i]->label, "RDREC") != 0) {*//첫번째 CSECT*

**if** (i == 0)*//첫번째 주소는 0*

tok\_adr[i] = 0;

tok\_adr[i] = locctr;*//증가된 locctr 저장*

**if**(token\_table[i]->operand[0] == NULL) {

**if** ((strcmp(token\_table[i]->operator, "LTORG")) == 0 || (strcmp(token\_table[i]->operator, "END")) == 0) {

literal\_table[lit\_line].addr = locctr;

locctr += 3;

lit\_line++;

}

}

**else** **if** (strncmp(token\_table[i]->operand[0], "=", 1) == 0) { *//리터럴이면*

strcpy(literal\_table[lit\_line].literal, token\_table[i]->operand[0]);*//리터럴테이블에 복사*

}

sym\_table[sym\_line].symbol[0] = "";

**if** (token\_table[i]->label != "") {

strcpy(sym\_table[sym\_line].symbol, token\_table[i]->label);*//label이 있으면 symbol에 저장*

}

**else**

strcpy(sym\_table[sym\_line].symbol, "t");

**if** (search\_opcode(token\_table[i]->operator) >= 0) {*//operator일 경우*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 1)*//1형식*

locctr += 1;

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2)*//2형식*

locctr += 2;

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {*//3or4*

**if** (token\_table[i]->operator[0] != '+') {*//3형식*

locctr += 3;

}

**else***//4형식*

locctr += 4;

}

}

**else** **if** (strcmp(token\_table[i]->operator, "RESW") == 0) {*//RESW*

temaddr = atoi(token\_table[i]->operand[0]);

locctr += (3 \* temaddr);

}

**else** **if** (strcmp(token\_table[i]->operator, "RESB") == 0) {

temaddr = atoi(token\_table[i]->operand[0]);

locctr += temaddr;

}

**else** **if** (strcmp(token\_table[i]->operator, "BYTE") == 0) {

locctr += 1;

}

**else** **if** (strcmp(token\_table[i]->operator, "EQU") == 0) {

**if** (strcmp(token\_table[i]->label, "MAXLEN") == 0) {

int adr1 = 0, adr2 = 0;

char\* ptr = strtok(token\_table[i]->operand[0], "-");

int j;

**if** (ptr != NULL) {

**for** (j = 0; j < sym\_line; j++) {

**if** (strcmp(sym\_table[j].symbol, ptr) == 0) {

adr1= sym\_table[j].addr;

**break**;

}

}

}

ptr = strtok(NULL, "-");

**if** (ptr != NULL) {

**for** (j = 0; j < sym\_line; j++) {

**if** (strcmp(sym\_table[j].symbol, ptr) == 0) {

adr2 = sym\_table[j].addr;

**break**;

}

}

}

tok\_adr[i] = adr1 - adr2;

}

}

**if** (sym\_table[sym\_line].symbol[0] != NULL) {

sym\_table[sym\_line].addr = tok\_adr[i];

}

i++;

sym\_line++;

len++;

}

csect\_len[0] = len - 1;

int k = 0;

locctr = 0;

len = 0;

**while** (strcmp(token\_table[i]->label, "WRREC") != 0) {*//두 번째 CSECT*

**if** (k == 0)*//첫번째 주소는 0*

tok\_adr[i] = 0;

tok\_adr[i] = locctr;*//증가된 locctr 저장*

**if** (token\_table[i]->operand[0] == NULL) {

**if** ((strcmp(token\_table[i]->operator, "LTORG")) == 0 || (strcmp(token\_table[i]->operator, "END")) == 0) {

literal\_table[lit\_line].addr = locctr;

locctr += 3;

lit\_line++;

}

}

**else** **if** (strncmp(token\_table[i]->operand[0], "=", 1) == 0) { *//리터럴이면*

strcpy(literal\_table[lit\_line].literal, token\_table[i]->operand[0]);*//리터럴테이블에 복사*

}

sym\_table[sym\_line].symbol[0] = "";

**if** (token\_table[i]->label != "") {

strcpy(sym\_table[sym\_line].symbol, token\_table[i]->label);*//label이 있으면 symbol에 저장*

}

**else**

strcpy(sym\_table[sym\_line].symbol, "t");

**if** (search\_opcode(token\_table[i]->operator) >= 0) {*//operator일 경우*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 1)*//1형식*

locctr += 1;

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2)*//2형식*

locctr += 2;

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {*//3or4*

**if** (token\_table[i]->operator[0] != '+') {*//3형식*

locctr += 3;

}

**else***//4형식*

locctr += 4;

}

}

**else** **if** (strcmp(token\_table[i]->operator, "RESW") == 0) {*//RESW*

temaddr = atoi(token\_table[i]->operand[0]);

locctr += (3 \* temaddr);

}

**else** **if** (strcmp(token\_table[i]->operator, "RESB") == 0) {

temaddr = atoi(token\_table[i]->operand[0]);

locctr += temaddr;

}

**else** **if** (strcmp(token\_table[i]->operator, "BYTE") == 0) {

locctr += 1;

}

**else** **if** (strcmp(token\_table[i]->operator, "WORD") == 0) {

locctr += 3;

}

**else** **if** (strcmp(token\_table[i]->operator, "EQU") == 0) {

**if** (strcmp(token\_table[i]->label, "MAXLEN") == 0) {

int adr1 = 0, adr2 = 0;

char\* ptr = strtok(token\_table[i]->operand[0], "-");

int j;

**if** (ptr != NULL) {

**for** (j = 0; j < sym\_line; j++) {

**if** (strcmp(sym\_table[j].symbol, ptr) == 0) {

adr1 = sym\_table[j].addr;

**break**;

}

}

}

ptr = strtok(NULL, "-");

**if** (ptr != NULL) {

**for** (j = 0; j < sym\_line; j++) {

**if** (strcmp(sym\_table[j].symbol, ptr) == 0) {

adr2 = sym\_table[j].addr;

**break**;

}

}

}

tok\_adr[i] = adr1 - adr2;

}

}

**if** (sym\_table[sym\_line].symbol[0] != NULL) {

sym\_table[sym\_line].addr = tok\_adr[i];

}

i++;

sym\_line++;

k++;

len++;

}

csect\_len[1] = len - 1;

locctr = 0;

k = 0;

len = 0;

**while** (i >= 0) {*//세 번째 CSECT*

**if** (k == 0)*//첫번째 주소는 0*

tok\_adr[i] = 0;

tok\_adr[i] = locctr;*//증가된 locctr 저장*

**if** ((token\_table[i]->operand[0] == NULL) || strcmp(token\_table[i]->operand[0], "FIRST") == 0) {

**if** ((strcmp(token\_table[i]->operator, "LTORG")) == 0 || (strcmp(token\_table[i]->operator, "END")) == 0) {

literal\_table[lit\_line].addr = locctr;

locctr += 3;

lit\_line++;

}

}

**else** **if** (strncmp(token\_table[i]->operand[0], "=", 1) == 0) { *//리터럴이면*

strcpy(literal\_table[lit\_line].literal, token\_table[i]->operand[0]);*//리터럴테이블에 복사*

}

sym\_table[sym\_line].symbol[0] = "";

**if** (token\_table[i]->label != "") {

strcpy(sym\_table[sym\_line].symbol, token\_table[i]->label);*//label이 있으면 symbol에 저장*

}

**else**

strcpy(sym\_table[sym\_line].symbol, "t");

**if** (search\_opcode(token\_table[i]->operator) >= 0) {*//operator일 경우*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 1)*//1형식*

locctr += 1;

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2)*//2형식*

locctr += 2;

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {*//3or4*

**if** (token\_table[i]->operator[0] != '+') {*//3형식*

locctr += 3;

}

**else***//4형식*

locctr += 4;

}

}

**else** **if** (strcmp(token\_table[i]->operator, "RESW") == 0) {*//RESW*

temaddr = atoi(token\_table[i]->operand[0]);

locctr += (3 \* temaddr);

}

**else** **if** (strcmp(token\_table[i]->operator, "RESB") == 0) {

temaddr = atoi(token\_table[i]->operand[0]);

locctr += temaddr;

}

**else** **if** (strcmp(token\_table[i]->operator, "BYTE") == 0) {

locctr += 1;

}

**else** **if** (strcmp(token\_table[i]->operator, "EQU") == 0) {

**if** (strcmp(token\_table[i]->label, "MAXLEN") == 0) {

int adr1 = 0, adr2 = 0;

char\* ptr = strtok(token\_table[i]->operand[0], "-");

int j;

**if** (ptr != NULL) {

**for** (j = 0; j < sym\_line; j++) {

**if** (strcmp(sym\_table[j].symbol, ptr) == 0) {

adr1 = sym\_table[j].addr;

**break**;

}

}

}

ptr = strtok(NULL, "-");

**if** (ptr != NULL) {

**for** (j = 0; j < sym\_line; j++) {

**if** (strcmp(sym\_table[j].symbol, ptr) == 0) {

adr2 = sym\_table[j].addr;

**break**;

}

}

}

tok\_adr[i] = adr1 - adr2;

}

}

**if** (sym\_table[sym\_line].symbol[0] != NULL) {

sym\_table[sym\_line].addr = tok\_adr[i];

}

**if** (strcmp(token\_table[i]->operator, "END") == 0) {

**break**;

}

i++;

sym\_line++;

len++;

}

csect\_len[2] = len - 1;

*/\* input\_data의 문자열을 한줄씩 입력 받아서*

*\* token\_parsing()을 호출하여 token\_unit에 저장*

*\*/*

**return** 0;

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.*

*\* 여기서 출력되는 내용은 SYMBOL별 주소값이 저장된 TABLE이다.*

*\* 매계 : 생성할 오브젝트 파일명*

*\* 반환 : 없음*

*\* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어*

*\* 화면에 출력해준다.*

*\**

*\* -----------------------------------------------------------------------------------*

*\*/*

void make\_symtab\_output(char \*file\_name)

{

*/\* add your code here \*/*

FILE\* file;

**if** (file\_name != NULL) {

file = fopen(file\_name, "w");

**for** (int i = 0; i < sym\_line; i++) {

**if** (sym\_table[i].symbol[0] != 't') {

**if**(strcmp(sym\_table[i].symbol, "RDREC") == 0 || (strcmp(sym\_table[i].symbol, "WRREC") == 0))

fprintf(file, "**\n**");

fprintf(file, "%s**\t\t**%X**\n**", sym\_table[i].symbol, sym\_table[i].addr);

}

}

}

**else** {

**for** (int i = 0; i < sym\_line; i++) {

**if** (sym\_table[i].symbol[0] != 't') {

**if** (strcmp(sym\_table[i].symbol, "RDREC") == 0 || (strcmp(sym\_table[i].symbol, "WRREC") == 0))

printf("**\n**");

printf("%s**\t\t**%X**\n**", sym\_table[i].symbol, sym\_table[i].addr);

}

}

printf("**\n**");

}

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.*

*\* 여기서 출력되는 내용은 LITERAL별 주소값이 저장된 TABLE이다.*

*\* 매계 : 생성할 오브젝트 파일명*

*\* 반환 : 없음*

*\* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어*

*\* 화면에 출력해준다.*

*\**

*\* -----------------------------------------------------------------------------------*

*\*/*

void make\_literaltab\_output(char \*file\_name)

{

*/\* add your code here \*/*

FILE\* file;

char\* str;

char str1[10];

**if** (file\_name != NULL) {

file = fopen(file\_name, "w");

**for** (int i = 0; i < lit\_line; i++) {

strcpy(str1, literal\_table[i].literal);

str = strtok(str1, "**\'**");

str = strtok(NULL, "**\'**");

fprintf(file, "%s**\t\t**%X**\n**", str, literal\_table[i].addr);

}

}

**else** {

**for** (int i = 0; i < lit\_line; i++) {

strcpy(str1, literal\_table[i].literal);

str = strtok(str1, "**\'**");

str = strtok(NULL, "**\'**");

printf("%s**\t\t**%X**\n**", str, literal\_table[i].addr);

}

printf("**\n**");

}

}

*/\* --------------------------------------------------------------------------------\**

*\* ------------------------- 추후 프로젝트에서 사용할 함수 --------------------------\**

*\* --------------------------------------------------------------------------------\*/*

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 어셈블리 코드를 기계어 코드로 바꾸기 위한 패스2 과정을 수행하는 함수이다.*

*\* 패스 2에서는 프로그램을 기계어로 바꾸는 작업은 라인 단위로 수행된다.*

*\* 다음과 같은 작업이 수행되어 진다.*

*\* 1. 실제로 해당 어셈블리 명령어를 기계어로 바꾸는 작업을 수행한다.*

*\* 매계 : 없음*

*\* 반환 : 정상종료 = 0, 에러발생 = < 0*

*\* 주의 :*

*\* -----------------------------------------------------------------------------------*

*\*/*

**static** int assem\_pass2(void)

{

*/\* add your code here \*/*

int opcode;

int address;

int i = 0;

**while** (strcmp(token\_table[i]->label, "RDREC") != 0) {*//첫번째 section*

objectcode[i] = 0;

**if** (search\_opcode(token\_table[i]->operator) >= 0) {*//instruction*

token\_table[i]->nixbpe = 0;

sscanf(inst\_table[(search\_opcode(token\_table[i]->operator))]->opcode, "%X", &opcode);*//int형 opcode에 코드 저장*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format != 2){*//format3,4*

**if** (strncmp(token\_table[i]->operand[0], "#", 1) == 0) *//immediate*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 4;

**else** **if**(strncmp(token\_table[i]->operand[0], "@", 1) == 0) *//indirect*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 5;

**else**

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 3 << 4;

**if** ((token\_table[i]->operand[1] != NULL) && (strcmp(token\_table[i]->operand[1], "X") == 0))*//index*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 3;

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 1;*//PC relative*

**if** (strcmp(token\_table[i]->operator, "RSUB") == 0) {*//RUSB의 경우는 예외처리*

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

objectcode[i] = opcode << 16 | token\_table[i]->nixbpe << 12;

i++;

**continue**;

}

*//addr 구하기*

char\* str;

int check = -1;

int a = 0;

int e = 0;

**if** (token\_table[i]->operand[0][0] == '#' || token\_table[i]->operand[0][0] == '@') {

str = token\_table[i]->operand[0] + 1;

**if** (token\_table[i]->operand[0][0] == '#')

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

}

**else**

str = token\_table[i]->operand[0];

**while**(strcmp(sym\_table[a].symbol, "RDREC") != 0){

**if** (strcmp(str, sym\_table[a].symbol) == 0)*//같은 CSECT symbol이면 check == a;*

check = a;

a++;

}

**if** (check != -1) {*//symbol이 있으면*

address = sym\_table[check].addr - tok\_adr[i+1];

**if** (address < 0)

address = 4096 - tok\_adr[i + 1] + sym\_table[check].addr;

}

**else** {*//symbol이 없으면*

**if** (str[0] >= '0' && str[0] <= '9') {*//immediate #number*

sscanf(str, "%d", &address);

}

**else** {

**for** (a = 0; a < sym\_line; a++) {

**if** (strcmp(str, sym\_table[a].symbol) == 0) {*//다른 CSECT symbol이면 check == a;*

check = a;

**break**;

}

}

**if** (check != -1) {*//다른 CSECT symbol*

address = 0;

}

**else** {

**for** (a = 0; a < lit\_line; a++) {

**if** (strcmp(str, literal\_table[a].literal) == 0) {*//리터럴이면*

address = literal\_table[a].addr - tok\_adr[i + 1];

**break**;

}

**else**

**return** -1;*//에러*

}

}

}

}

**if**(strncmp(token\_table[i]->operator, "+", 1) != 0)

objectcode[i] = opcode << 16 | token\_table[i]->nixbpe << 12 | address;

**else** {

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1;

objectcode[i] = opcode << 24 | token\_table[i]->nixbpe << 20 | address;

}

}

**else** {*//2형식*

**if** (strcmp(token\_table[i]->operand[0], "X") == 0)

address = 16;

**else** **if** (strcmp(token\_table[i]->operand[0], "A") == 0)

address = 0;

**else** **if** (strcmp(token\_table[i]->operand[0], "S") == 0)

address = 64;

**else** **if** (strcmp(token\_table[i]->operand[0], "T") == 0)

address = 80;

objectcode[i] = opcode << 8 | address;

}

}

i++;

}

int k = i;

**while** (strcmp(token\_table[i]->label, "WRREC") != 0) {*//2번째 section*

objectcode[i] = 0;

**if** (search\_opcode(token\_table[i]->operator) >= 0) {*//instruction*

token\_table[i]->nixbpe = 0;

sscanf(inst\_table[(search\_opcode(token\_table[i]->operator))]->opcode, "%X", &opcode);*//int형 opcode에 코드 저장*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format != 2) {*//format3,4*

**if** (strncmp(token\_table[i]->operand[0], "#", 1) == 0) *//immediate*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 4;

**else** **if** (strncmp(token\_table[i]->operand[0], "@", 1) == 0) *//indirect*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 5;

**else**

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 3 << 4;

**if** ((token\_table[i]->operand[1] != NULL) && (strcmp(token\_table[i]->operand[1], "X") == 0))*//index*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 3;

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 1;*//PC relative*

**if** (strcmp(token\_table[i]->operator, "RSUB") == 0) {*//RUSB의 경우는 예외처리*

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

objectcode[i] = opcode << 16 | token\_table[i]->nixbpe << 12;

i++;

**continue**;

}

*//addr 구하기*

char\* str;

int check = -1;

int a = k;

**if** (token\_table[i]->operand[0][0] == '#' || token\_table[i]->operand[0][0] == '@'){

str = token\_table[i]->operand[0] + 1;

**if** (token\_table[i]->operand[0][0] == '#')

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

}

**else**

str = token\_table[i]->operand[0];

**while** (strcmp(sym\_table[a].symbol, "WRREC") != 0) {

**if** (strcmp(str, sym\_table[a].symbol) == 0)*//같은 CSECT symbol이면 check == a;*

check = a;

a++;

}

**if** (check != -1) {*//symbol이 있으면*

address = sym\_table[check].addr - tok\_adr[i + 1];

**if** (address < 0)

address = 4096 - tok\_adr[i + 1] + sym\_table[check].addr;

}

**else** {*//symbol이 없으면*

**if** (str[0] >= '0' && str[0] <= '9') {*//immediate #number*

sscanf(str, "%d", &address);

}

**else** {

**for** (a = 0; a < sym\_line; a++) {

**if** (strcmp(str, sym\_table[a].symbol) == 0) {*//다른 CSECT symbol이면 check == a;*

check = a;

**break**;

}

}

**if** (check != -1) {*//다른 CSECT symbol*

address = 0;

}

**else** {

**for** (a = 0; a < lit\_line; a++) {

**if** (strcmp(str, literal\_table[a].literal) == 0) {*//리터럴이면*

address = literal\_table[a].addr - tok\_adr[i + 1];

**break**;

}

}

}

}

}

**if** (strncmp(token\_table[i]->operator, "+", 1) != 0)

objectcode[i] = opcode << 16 | token\_table[i]->nixbpe << 12 | address;

**else** {

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1;

objectcode[i] = opcode << 24 | token\_table[i]->nixbpe << 20 | address;

}

}

**else** {*//2형식*

**if** (token\_table[i]->operand[0] != NULL) {

**if** (strcmp(token\_table[i]->operand[0], "X") == 0)

address = 1;

**else** **if** (strcmp(token\_table[i]->operand[0], "A") == 0)

address = 0;

**else** **if** (strcmp(token\_table[i]->operand[0], "S") == 0)

address = 4;

**else** **if** (strcmp(token\_table[i]->operand[0], "T") == 0)

address = 5;

objectcode[i] = opcode << 8 | address << 4;

}

**if** ((token\_table[i]->operand[0] != NULL) && (token\_table[i]->operand[1] != NULL)) {

**if** (strcmp(token\_table[i]->operand[1], "X") == 0)

address = 1;

**else** **if** (strcmp(token\_table[i]->operand[1], "A") == 0)

address = 0;

**else** **if** (strcmp(token\_table[i]->operand[1], "S") == 0)

address = 4;

**else** **if** (strcmp(token\_table[i]->operand[1], "T") == 0)

address = 5;

objectcode[i] = objectcode[i] | address;

}

}

}

i++;

}

k = i;

**while** (i >= 0) {*//3번째 section*

objectcode[i] = 0;

**if** (search\_opcode(token\_table[i]->operator) >= 0) {*//instruction*

token\_table[i]->nixbpe = 0;

sscanf(inst\_table[(search\_opcode(token\_table[i]->operator))]->opcode, "%X", &opcode);*//int형 opcode에 코드 저장*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format != 2) {*//format3,4*

**if** (strncmp(token\_table[i]->operand[0], "#", 1) == 0) *//immediate*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 4;

**else** **if** (strncmp(token\_table[i]->operand[0], "@", 1) == 0) *//indirect*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 5;

**else**

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 3 << 4;

**if** ((token\_table[i]->operand[1] != NULL) && (strcmp(token\_table[i]->operand[1], "X") == 0))*//index*

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 3;

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1 << 1;*//PC relative*

**if** (strcmp(token\_table[i]->operator, "RSUB") == 0) {*//RUSB의 경우는 예외처리*

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

objectcode[i] = opcode << 16 | token\_table[i]->nixbpe << 12;

i++;

**continue**;

}

*//address 구하기*

char\* str;

int check = -1;

int a = k;

**if** (token\_table[i]->operand[0][0] == '#' || token\_table[i]->operand[0][0] == '@'){

str = token\_table[i]->operand[0] + 1;

**if** (token\_table[i]->operand[0][0] == '#')

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

}

**else**

str = token\_table[i]->operand[0];

**while** (a <= sym\_line) {

**if** (strcmp(str, sym\_table[a].symbol) == 0)*//같은 CSECT symbol이면 check == a;*

check = a;

a++;

}

**if** (check != -1) {*//symbol이 있으면*

address = sym\_table[check].addr - tok\_adr[i + 1];

**if** (address < 0)

address = 4096 - tok\_adr[i + 1] + sym\_table[check].addr;

}

**else** {*//symbol이 없으면*

**if** (str[0] >= '0' && str[0] <= '9') {*//immediate #number*

sscanf(str, "%d", &address);

}

**else** {

**for** (a = 0; a < sym\_line; a++) {

**if** (strcmp(str, sym\_table[a].symbol) == 0) {*//다른 CSECT symbol이면 check == a;*

check = a;

**break**;

}

}

**if** (check != -1) {*//다른 CSECT symbol*

address = 0;

}

**else** {

**for** (a = 0; a < lit\_line; a++) {

**if** (strcmp(str, literal\_table[a].literal) == 0) {*//리터럴이면*

address = literal\_table[a].addr - tok\_adr[i + 1];

**break**;

}

}

}

}

}

**if** (strncmp(token\_table[i]->operator, "+", 1) != 0)

objectcode[i] = opcode << 16 | token\_table[i]->nixbpe << 12 | address;

**else** {

token\_table[i]->nixbpe = token\_table[i]->nixbpe & (~(1 << 1));

token\_table[i]->nixbpe = token\_table[i]->nixbpe | 1;

objectcode[i] = opcode << 24 | token\_table[i]->nixbpe << 20 | address;

}

}

**else** {*//2형식*

**if** (token\_table[i]->operand[0] != NULL) {

**if** (strcmp(token\_table[i]->operand[0], "X") == 0)

address = 1;

**else** **if** (strcmp(token\_table[i]->operand[0], "A") == 0)

address = 0;

**else** **if** (strcmp(token\_table[i]->operand[0], "S") == 0)

address = 4;

**else** **if** (strcmp(token\_table[i]->operand[0], "T") == 0)

address = 5;

objectcode[i] = opcode << 8 | address << 4;

}

**if** ((token\_table[i]->operand[0] != NULL) && (token\_table[i]->operand[1] != NULL)) {

**if** (strcmp(token\_table[i]->operand[1], "X") == 0)

address = 1;

**else** **if** (strcmp(token\_table[i]->operand[1], "A") == 0)

address = 0;

**else** **if** (strcmp(token\_table[i]->operand[1], "S") == 0)

address = 4;

**else** **if** (strcmp(token\_table[i]->operand[1], "T") == 0)

address = 5;

objectcode[i] = objectcode[i] | address;

}

}

}

**if** (strcmp(token\_table[i]->operator, "END") == 0) {

**break**;

}

i++;

}

**return** 0;

}

*/\* ----------------------------------------------------------------------------------*

*\* 설명 : 입력된 문자열의 이름을 가진 파일에 프로그램의 결과를 저장하는 함수이다.*

*\* 여기서 출력되는 내용은 object code (프로젝트 1번) 이다.*

*\* 매계 : 생성할 오브젝트 파일명*

*\* 반환 : 없음*

*\* 주의 : 만약 인자로 NULL값이 들어온다면 프로그램의 결과를 표준출력으로 보내어*

*\* 화면에 출력해준다.*

*\**

*\* -----------------------------------------------------------------------------------*

*\*/*

void make\_objectcode\_output(char\* file\_name)

{

*/\* add your code here \*/*

FILE\* file;

**if** (file\_name != NULL) {

file = fopen(file\_name, "w");

int i = 0;

int txtlen1 = 0;*//txt길이변수*

int txtlen2 = 0;

int start = 0;*//명령어 시작위치*

int second = 0;*//두번째 T 시작위치*

**while** (strcmp(token\_table[i]->label, "RDREC") != 0) {*//section 1*

fprintf(file, "H%s**\t**%6s%06X**\n**", token\_table[i]->label, "000000", tok\_adr[csect\_len[0] - 1]);

**for** (i = i + 1; i < i + 3; i++) { *//EXTDEF, EXTREF 처리*

**if** (strcmp(token\_table[i]->operator, "EXTDEF") == 0) {

fputc('D', file);

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

fprintf(file, "%s", token\_table[i]->operand[j]);

fprintf(file, "%06X", sym\_table[search\_symbol(token\_table[i]->operand[j])].addr);

}

fputc('\n', file);

}

**else** **if** (strcmp(token\_table[i]->operator, "EXTREF") == 0) {

fputc('R', file);

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

fprintf(file, "%s", token\_table[i]->operand[j]);

}

fputc('\n', file);

start = i + 1;

**break**;

}

}

*//instruction시작*

int cnt1 = 0;*//mod배열 count*

int j;

i = start;

**while** ((txtlen1 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//첫번째 T length측정(BYTE단위)*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {*//2형식 +2*

**if** (txtlen1 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {*//4형식 +4*

**if** (txtlen1 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 4;

}

**else** {*//3형식 +3*

**if** (txtlen1 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "RDREC") == 0 || strcmp(token\_table[i]->operand[0], "WRREC") == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1;

cnt1++;

}

i++;

}

fprintf(file, "T%06X%02X", tok\_adr[start], txtlen1);

**for** (j = start; j < i; j++) {*//첫번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

fprintf(file, "%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

fprintf(file, "%08X", objectcode[j]);

**else**

fprintf(file, "%06X", objectcode[j]);

}

}

second = i;

fputc('\n', file);

**while** ((txtlen2 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//두번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen2 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen2 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 4;

}

**else** {

**if** (txtlen2 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "RDREC") == 0 || strcmp(token\_table[i]->operand[0], "WRREC") == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1 + txtlen2;

cnt1++;

}

i++;

}

fprintf(file, "T%06X%02X", tok\_adr[j], txtlen2);

**for** (j = second; j < i; j++) {*//두번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

fprintf(file, "%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

fprintf(file, "%08X", objectcode[j]);

**else**

fprintf(file, "%06X", objectcode[j]);

}

}

fputc('\n', file);

cnt1 = 0;

**for** (j = start; j < i; j++) {*//modify출력*

**if** (strcmp(token\_table[j]->operand[0], "RDREC") == 0) {

fprintf(file, "M%06X05+RDREC**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "WRREC") == 0) {

fprintf(file, "M%06X05+WRREC**\n**", mod[cnt1] - 3);

cnt1++;

}

}

fprintf(file, "E000000**\n\n**");

**break**;

}

*//section 2*

txtlen1 = 0;

txtlen2 = 0;

start = 0;

second = 0;

**while** (strcmp(token\_table[i]->label, "WRREC") != 0) {

i = csect\_len[0] + 1;

fprintf(file, "H%s**\t**%6s%06X**\n**", token\_table[i]->label, "000000", tok\_adr[csect\_len[0]+csect\_len[1] + 1]+3);

**for** (i = i + 1; i < i + 3; i++) { *//EXTDEF, EXTREF 처리*

**if** (strcmp(token\_table[i]->operator, "EXTDEF") == 0) {

fputc('D', file);

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

fprintf(file, "%s", token\_table[i]->operand[j]);

fprintf(file, "%06X", sym\_table[search\_symbol(token\_table[i]->operand[j])].addr);

}

fputc('\n', file);

}

**else** **if** (strcmp(token\_table[i]->operator, "EXTREF") == 0) {

fputc('R', file);

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

fprintf(file, "%s", token\_table[i]->operand[j]);

}

fputc('\n', file);

start = i + 1;

**break**;

}

}

*//instruction시작*

int cnt1 = 0;*//mod배열 count*

int j;

i = start;

**while** ((txtlen1 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//첫번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen1 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen1 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 4;

}

**else** {

**if** (txtlen1 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "BUFFER") == 0 || strcmp(token\_table[i]->operand[0], "LENGTH") == 0 || strncmp(token\_table[i]->operand[0], "BUFEND", 6) == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1 + txtlen2;

cnt1++;

}

i++;

}

fprintf(file, "T%06X%02X", tok\_adr[start], txtlen1);

**for** (j = start; j < i; j++) {*//첫번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

fprintf(file, "%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

fprintf(file, "%08X", objectcode[j]);

**else**

fprintf(file, "%06X", objectcode[j]);

}

}

second = i;

fputc('\n', file);

**while** ((txtlen2 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//두번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen2 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen2 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 4;

}

**else** {

**if** (txtlen2 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "BUFFER") == 0 || strcmp(token\_table[i]->operand[0], "LENGTH") == 0 || strncmp(token\_table[i]->operand[0], "BUFEND", 6) == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1 + txtlen2;

cnt1++;

}

i++;

}

**if** (strcmp(token\_table[i]->operator, "BYTE") == 0) {*//X'F1' 처리*

char\* str1;

char str[10];

strcpy(str, token\_table[i]->operand[0]);

str1 = strtok(str, "**\'**");

str1 = strtok(NULL, "**\'**");

print[i] = strtol(str1, NULL, 16);

txtlen2 += 1;

i++;

}

**if** (strcmp(token\_table[i]->operator, "WORD") == 0) {*//MAXLEN 처리*

print[i] = 0;

txtlen2 += 3;

i++;

}

fprintf(file, "T%06X%02X", tok\_adr[j], txtlen2);

**for** (j = second; j < i; j++) {*//두번째T 출력*

**if** (strcmp(token\_table[j]->operator, "BYTE") == 0)

fprintf(file, "%02X", print[j]);

**else** **if** (strcmp(token\_table[j]->operator, "WORD") == 0)

fprintf(file, "%06X", print[j]);

**else** **if** (search\_opcode(token\_table[j]->operator) >= 0 && inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

fprintf(file, "%04X", print[j]);

**else** **if** (search\_opcode(token\_table[j]->operator) >= 0 && inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

fprintf(file, "%08X", objectcode[j]);

**else**

fprintf(file, "%06X", objectcode[j]);

}

}

fputc('\n', file);

cnt1 = 0;

**for** (j = start; j < i; j++) {*//modify출력*

**if** (strcmp(token\_table[j]->operand[0], "BUFFER") == 0) {

fprintf(file, "M%06X05+BUFFER**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "LENGTH") == 0) {

fprintf(file, "M%06X05+LENGTH**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "BUFEND-BUFFER") == 0) {

char\* str2;

char str3[15];

strcpy(str3, token\_table[j]->operand[0]);

str2 = strtok(str3, "\-");

fprintf(file, "M%06X06+%s**\n**", mod[cnt1] - 3, str3);

str2 = strtok(NULL, "\-");

fprintf(file, "M%06X06+%s**\n**", mod[cnt1] - 3, str2);

cnt1++;

}

}

fprintf(file, "E**\n\n**");

**break**;

}

*//section 3*

txtlen1 = 0;

txtlen2 = 0;

start = 0;

second = 0;

**while** (strcmp(token\_table[i]->operator, "END") != 0) {

i = csect\_len[0] + csect\_len[1]+ 2;

fprintf(file, "H%s**\t**%6s%06X**\n**", token\_table[i]->label, "000000", tok\_adr[i+csect\_len[2]+1] + 1);

**for** (i = i + 1; i < i + 3; i++) { *//EXTDEF, EXTREF 처리*

**if** (strcmp(token\_table[i]->operator, "EXTDEF") == 0) {

fputc('D', file);

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

fprintf(file, "%s", token\_table[i]->operand[j]);

fprintf(file, "%06X", sym\_table[search\_symbol(token\_table[i]->operand[j])].addr);

}

fputc('\n', file);

}

**else** **if** (strcmp(token\_table[i]->operator, "EXTREF") == 0) {

fputc('R', file);

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

fprintf(file, "%s", token\_table[i]->operand[j]);

}

fputc('\n', file);

start = i + 1;

**break**;

}

}

*//instruction시작*

int cnt1 = 0;*//mod배열 count*

int j;

i = start;

**while** ((txtlen1 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//첫번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen1 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen1 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 4;

}

**else** {

**if** (txtlen1 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 3;

}

}

**if** (strcmp(token\_table[i+1]->operator, "END") == 0)

txtlen1 += 1;

**if** (strcmp(token\_table[i]->operand[0], "BUFFER") == 0 || strcmp(token\_table[i]->operand[0], "LENGTH") == 0 || strncmp(token\_table[i]->operand[0], "BUFEND", 6) == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1;

cnt1++;

}

i++;

}

fprintf(file, "T%06X%02X", tok\_adr[start], txtlen1);

**for** (j = start; j < i; j++) {*//첫번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

fprintf(file, "%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

fprintf(file, "%08X", objectcode[j]);

**else**

fprintf(file, "%06X", objectcode[j]);

}

**if** (strcmp(token\_table[j + 1]->operator, "END") == 0) {

char\* str;

char str1[10];

strcpy(str1, literal\_table[1].literal);

str = strtok(str1, "**\'**");

str = strtok(NULL, "**\'**");

fprintf(file, "%s", str);

}

}

fputc('\n', file);

cnt1 = 0;

**for** (j = start; j < i; j++) {*//modify출력*

**if** (strcmp(token\_table[j]->operand[0], "BUFFER") == 0) {

fprintf(file, "M%06X05+BUFFER**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "LENGTH") == 0) {

fprintf(file, "M%06X05+LENGTH**\n**", mod[cnt1] - 3);

cnt1++;

}

}

fprintf(file, "E**\n\n**");

**break**;

}

}

**else** {*//file\_name == NULL일때, 이하 내용 출력방식(printf, putchar)외에 같음*

int i = 0;

int txtlen1 = 0;*//txt길이변수*

int txtlen2 = 0;

int start = 0;*//명령어 시작위치*

int second = 0;*//두번째 T 시작위치*

**while** (strcmp(token\_table[i]->label, "RDREC") != 0) {*//section 1*

printf("H%s**\t**%6s%06X**\n**", token\_table[i]->label, "000000", tok\_adr[csect\_len[0] - 1]);

**for** (i = i + 1; i < i + 3; i++) { *//EXTDEF, EXTREF 처리*

**if** (strcmp(token\_table[i]->operator, "EXTDEF") == 0) {

putchar('D');

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

printf("%s", token\_table[i]->operand[j]);

printf("%06X", sym\_table[search\_symbol(token\_table[i]->operand[j])].addr);

}

putchar('\n');

}

**else** **if** (strcmp(token\_table[i]->operator, "EXTREF") == 0) {

putchar('R');

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

printf("%s", token\_table[i]->operand[j]);

}

putchar('\n');

start = i + 1;

**break**;

}

}

*//instruction시작*

int cnt1 = 0;*//mod배열 count*

int j;

i = start;

**while** ((txtlen1 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//첫번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen1 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen1 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 4;

}

**else** {

**if** (txtlen1 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "RDREC") == 0 || strcmp(token\_table[i]->operand[0], "WRREC") == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1;

cnt1++;

}

i++;

}

printf("T%06X%02X", tok\_adr[start], txtlen1);

**for** (j = start; j < i; j++) {*//첫번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

printf("%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

printf("%08X", objectcode[j]);

**else**

printf("%06X", objectcode[j]);

}

}

second = i;

putchar('\n');

**while** ((txtlen2 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//두번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen2 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen2 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 4;

}

**else** {

**if** (txtlen2 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "RDREC") == 0 || strcmp(token\_table[i]->operand[0], "WRREC") == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1 + txtlen2;

cnt1++;

}

i++;

}

printf("T%06X%02X", tok\_adr[j], txtlen2);

**for** (j = second; j < i; j++) {*//두번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

printf("%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

printf("%08X", objectcode[j]);

**else**

printf("%06X", objectcode[j]);

}

}

putchar('\n');

cnt1 = 0;

**for** (j = start; j < i; j++) {*//modify출력*

**if** (strcmp(token\_table[j]->operand[0], "RDREC") == 0) {

printf("M%06X05+RDREC**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "WRREC") == 0) {

printf("M%06X05+WRREC**\n**", mod[cnt1] - 3);

cnt1++;

}

}

printf("E000000**\n\n**");

**break**;

}

*//section 2*

txtlen1 = 0;

txtlen2 = 0;

start = 0;

second = 0;

**while** (strcmp(token\_table[i]->label, "WRREC") != 0) {

i = csect\_len[0] + 1;

printf("H%s**\t**%6s%06X**\n**", token\_table[i]->label, "000000", tok\_adr[csect\_len[0] + csect\_len[1] + 1] + 3);

**for** (i = i + 1; i < i + 3; i++) { *//EXTDEF, EXTREF 처리*

**if** (strcmp(token\_table[i]->operator, "EXTDEF") == 0) {

putchar('D');

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

printf("%s", token\_table[i]->operand[j]);

printf("%06X", sym\_table[search\_symbol(token\_table[i]->operand[j])].addr);

}

putchar('\n');

}

**else** **if** (strcmp(token\_table[i]->operator, "EXTREF") == 0) {

putchar('R');

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

printf("%s", token\_table[i]->operand[j]);

}

putchar('\n');

start = i + 1;

**break**;

}

}

*//instruction시작*

int cnt1 = 0;*//mod배열 count*

int j;

i = start;

**while** ((txtlen1 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//첫번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen1 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen1 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 4;

}

**else** {

**if** (txtlen1 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "BUFFER") == 0 || strcmp(token\_table[i]->operand[0], "LENGTH") == 0 || strncmp(token\_table[i]->operand[0], "BUFEND", 6) == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1 + txtlen2;

cnt1++;

}

i++;

}

printf("T%06X%02X", tok\_adr[start], txtlen1);

**for** (j = start; j < i; j++) {*//첫번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

printf("%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

printf("%08X", objectcode[j]);

**else**

printf("%06X", objectcode[j]);

}

}

second = i;

putchar('\n');

**while** ((txtlen2 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//두번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen2 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen2 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 4;

}

**else** {

**if** (txtlen2 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen2 += 3;

}

}

**if** (strcmp(token\_table[i]->operand[0], "BUFFER") == 0 || strcmp(token\_table[i]->operand[0], "LENGTH") == 0 || strncmp(token\_table[i]->operand[0], "BUFEND", 6) == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1 + txtlen2;

cnt1++;

}

i++;

}

**if** (strcmp(token\_table[i]->operator, "BYTE") == 0) {

char\* str1;

char str[10];

strcpy(str, token\_table[i]->operand[0]);

str1 = strtok(str, "**\'**");

str1 = strtok(NULL, "**\'**");

print[i] = strtol(str1, NULL, 16);

txtlen2 += 1;

i++;

}

**if** (strcmp(token\_table[i]->operator, "WORD") == 0) {

print[i] = 0;

txtlen2 += 3;

i++;

}

printf("T%06X%02X", tok\_adr[j], txtlen2);

**for** (j = second; j < i; j++) {*//두번째T 출력*

**if** (strcmp(token\_table[j]->operator, "BYTE") == 0)

printf("%02X", print[j]);

**else** **if** (strcmp(token\_table[j]->operator, "WORD") == 0)

printf("%06X", print[j]);

**else** **if** (search\_opcode(token\_table[j]->operator) >= 0 && inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

printf("%04X", print[j]);

**else** **if** (search\_opcode(token\_table[j]->operator) >= 0 && inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

printf("%08X", objectcode[j]);

**else**

printf("%06X", objectcode[j]);

}

}

putchar('\n');

cnt1 = 0;

**for** (j = start; j < i; j++) {*//modify출력*

**if** (strcmp(token\_table[j]->operand[0], "BUFFER") == 0) {

printf("M%06X05+BUFFER**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "LENGTH") == 0) {

printf("M%06X05+LENGTH**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "BUFEND-BUFFER") == 0) {

char\* str2;

char str3[15];

strcpy(str3, token\_table[j]->operand[0]);

str2 = strtok(str3, "\-");

printf("M%06X06+%s**\n**", mod[cnt1] - 3, str3);

str2 = strtok(NULL, "\-");

printf("M%06X06+%s**\n**", mod[cnt1] - 3, str2);

cnt1++;

}

}

printf("E**\n\n**");

**break**;

}

*//section 3*

txtlen1 = 0;

txtlen2 = 0;

start = 0;

second = 0;

**while** (strcmp(token\_table[i]->operator, "END") != 0) {

i = csect\_len[0] + csect\_len[1] + 2;

printf("H%s**\t**%6s%06X**\n**", token\_table[i]->label, "000000", tok\_adr[i + csect\_len[2] + 1] + 1);

**for** (i = i + 1; i < i + 3; i++) { *//EXTDEF, EXTREF 처리*

**if** (strcmp(token\_table[i]->operator, "EXTDEF") == 0) {

putchar('D');

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

printf("%s", token\_table[i]->operand[j]);

printf("%06X", sym\_table[search\_symbol(token\_table[i]->operand[j])].addr);

}

putchar('\n');

}

**else** **if** (strcmp(token\_table[i]->operator, "EXTREF") == 0) {

putchar('R');

**for** (int j = 0; j < MAX\_OPERAND; j++) {

**if** (token\_table[i]->operand[j] == NULL)

**break**;

printf("%s", token\_table[i]->operand[j]);

}

putchar('\n');

start = i + 1;

**break**;

}

}

*//instruction시작*

int cnt1 = 0;*//mod배열 count*

int j;

i = start;

**while** ((txtlen1 <= 30) && search\_opcode(token\_table[i]->operator) >= 0) {*//첫번째 T length측정*

**if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 2) {

**if** (txtlen1 + 2 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 2;

}

**else** **if** (inst\_table[search\_opcode(token\_table[i]->operator)]->format == 3) {

**if** (strncmp(token\_table[i]->operator, "+", 1) == 0) {

**if** (txtlen1 + 4 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 4;

}

**else** {

**if** (txtlen1 + 3 > 30)

**break**;

print[i] = objectcode[i];

txtlen1 += 3;

}

}

**if** (strcmp(token\_table[i + 1]->operator, "END") == 0)

txtlen1 += 1;

**if** (strcmp(token\_table[i]->operand[0], "BUFFER") == 0 || strcmp(token\_table[i]->operand[0], "LENGTH") == 0 || strncmp(token\_table[i]->operand[0], "BUFEND", 6) == 0) {*//modify위치 저장*

mod[cnt1] = txtlen1;

cnt1++;

}

i++;

}

printf("T%06X%02X", tok\_adr[start], txtlen1);

**for** (j = start; j < i; j++) {*//첫번째T 출력*

**if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 2)

printf("%04X", print[j]);

**else** **if** (inst\_table[search\_opcode(token\_table[j]->operator)]->format == 3) {

**if** (strncmp(token\_table[j]->operator, "+", 1) == 0)

printf("%08X", objectcode[j]);

**else**

printf("%06X", objectcode[j]);

}

**if** (strcmp(token\_table[j + 1]->operator, "END") == 0) {

char\* str;

char str1[10];

strcpy(str1, literal\_table[1].literal);

str = strtok(str1, "**\'**");

str = strtok(NULL, "**\'**");

printf("%s", str);

}

}

putchar('\n');

cnt1 = 0;

**for** (j = start; j < i; j++) {*//modify출력*

**if** (strcmp(token\_table[j]->operand[0], "BUFFER") == 0) {

printf("M%06X05+BUFFER**\n**", mod[cnt1] - 3);

cnt1++;

}

**else** **if** (strcmp(token\_table[j]->operand[0], "LENGTH") == 0) {

printf("M%06X05+LENGTH**\n**", mod[cnt1] - 3);

cnt1++;

}

}

printf("E**\n\n**");

**break**;

}

}

}

int search\_symbol(char\* str)*//입력받은 string이 sym\_table에 있는지 확인, 있다면 해당 line number 반환*

{

int i;

**for** (i = 0; i < sym\_line; i++) {

**if** (strcmp(str, sym\_table[i].symbol) == 0) {

**return** i;

}

}

**return** -1;

}